Tentamen/Exam

TDDB44 Kompilatorkonstruktion / Compiler Construction

TDDD55 Kompilatorer och interpretatorer / Compilers and Interpreters

2018-01-10, 08:00 – 12:00

Hjälpmedel / Allowed material:

- Engelsk ordbok / Dictionary from/to English to/from your native language
- Miniräknare / Pocket calculator

General instructions:

- Read the instructions and examination procedures for exams at LiU.
- Read all assignments carefully and completely before you begin.
- Note that not every problem is for all courses. Watch out for comments like "TDDD55 only".
- You may answer in Swedish or in English.
- Write clearly – unreadable text will be ignored. Be precise in your statements – imprecise formulations may lead to reduction of points. Motivate clearly all statements and reasoning. Explain calculations and solution procedures.
- The assignments are not ordered according to difficulty.
- The exam is designed for 40 points (per course). You may thus plan 6 minutes per point.
- Grading: U, 3, 4, 5 resp. Fx, C, B, A.
- The preliminary threshold for passing (grade 3/C) is 20 points.

1. (TDDD55 only – 6p) **Formal Languages and Automata Theory**

   Consider the language L consisting of all strings w over the alphabet $\{0, 1\}$ such that every string which contain 00 also contain 11. Example of strings in the language: 0, 1, 11, 1101001001101, 1001011. Examples of strings not in the language: 101001, 0010.

   (a) (1.5p) Construct a regular expression for L.

   (b) (1.5p) Construct from the regular expression an NFA recognizing L.

   (c) (2.5p) Construct a DFA recognizing L, either by deriving it from the NFA or by constructing it directly.

   (d) (0.5p) Give an example of a formal language that is not context-free.

2. (3p) **Compiler Structure and Generators**

   (a) (2p) What are the generated compiler phases and what are the corresponding formalisms (mention at least 5 phases and 3 formalisms) when using a compiler generator to generate a compiler?

   (b) (1p) Most modern compilers have not just one, but several intermediate representations (IR). What is the advantage of having more than one IR, and what could be the drawback?

3. (3p) **Symbol Table Management**

   Describe what the compiler – using a symbol table implemented as a hash table with chaining and block scoped control – does in compiling a statically scoped, block structured language when it handles:

   (a) (0.5p) block entry

   (b) (1.0p) block exit

   (c) (0.5p) a variable declaration

   (d) (1.0p) a variable use.

4. (5p) **Top-Down Parsing**

Given a grammar with nonterminals X, Y, Z, and S, where S is the start symbol, and the following productions:

```
1. S ::= X ^ X
2. X ::= X * Y
3.     | Y
4. Y ::= Z + Y
5.     | Z
6. Z ::= a
7.     | b
```

Assume that ^, *, and + are operators.

(a) (1p) What is the associativity (right, left, none) of the operators?

(b) (1p) What is the precedence (relative priority) between the operators?

(c) (2p) Can the grammar be used directly for a recursive-descent parser? Motivate your answer. If not, rewrite the grammar so that the language it defines can be parsed using the recursiuve-descent method.

(d) (2p) Write a recursive-descent parser to analyze the language defined by the grammar. (Pseudocode/program code without declarations is fine. Use the function scan() to read the next input token, and the function error() to report errors if needed.)

5. (TDDD55 only – 6p) **LR parsing**

(a) (3p) Use the SLR(1) tables below to show how the string `1-2-1*2` is parsed. You should show, step by step, how stack, input data etc. are changed during the parsing. Start state is `00`, start symbol is `S`.

Grammar:

```
1.  S ::= A + A
2.  A ::= B - A
3.      | B
4.  B ::= B * C
5.      | C
6.  C ::= 1
7.      | 2
```

Tables:

	Action						GOTO			
State	$	+	-	*	1	2	S	A	B	C
00	*	*	*	*	S09	S10	01	02	05	08
01	A	*	*	*	*	*	*	*	*	*
02	*	S03	*	*	*	*	*	*	*	*
03	*	*	*	*	S09	S10	*	04	05	08
04	R1	*	*	*	*	*	*	*	*	*
05	R3	R3	S06	S11	*	*	*	*	*	*
06	*	*	*	*	S09	S10	*	07	05	08
07	R2	R2	*	*	*	*	*	*	*	*
08	R5	R5	R5	R5	*	*	*	*	*	*
09	R6	R6	R6	R6	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	*	*	*	*	S09	S10	*	*	*	12
12	R4	R4	R4	R4	*	*	*	*	*	*

(b) (3p) The tables above are SLR(1) tables. Explain what the difference is from LR(0) tables. Show how to transform the tables into LR(0) tables or explain why this is not possible without rewriting the grammar.

6. (TDDB44 only – 6p) **LR parsing**

Given the following grammar G for strings over the alphabet {1,2,3,4} with nonterminals S, A and B, where S is the start symbol:

```
1. S ::= A
2. A ::= 1 A 2
3.     | 1 B 1
4.     | 1 B 3
5. B ::= 1 A 2
6.     | 2 A 4
7.     | 4
```

Is the grammar G in SLR(1) or even LR(0)? Justify your answer using the LR item sets. If it is: construct the characteristic LR-items NFA, the corresponding GOTO graph, the ACTION table and the GOTO table and show with tables and stack how the string 1214311 is parsed.

If it is not: describe where/how the problem occurs.

7. (5p) **Syntax-Directed Translation**

The Pascal language has a `break` statement that works similar to the following grammar:

```
<while_loop>  ::= while <expression> do begin <stmt_list> end semicolon
<stmt_list> ::= <stmt_list><stmt> |
<stmt>      ::= ... | break semicolon
```

(where "..." represents all other possible kinds of statements). `break` means that execution directly jumps to the exit of the enclosing loop.

Example:

```
while i<10 do
  begin
    i := i+1;
    if i>5 then
      break; (* Jump to L1 *)
    writeln(i);
  end;
L1: ...
```

Write a syntax-directed translation scheme, with attributes and semantic rules, for translating statements with `break` inside them, to quadruples. Note that `break` may only appear inside loop constructs; report errors using `error()`. The translation scheme should be used during bottom-up parsing. You are allowed to define and use symbolic labels. You may need to rewrite the grammar. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions. (Since it is a syntax-directed translation scheme, not an attribute grammar, generation of a quadruple puts it in an array of quadruples and attribute values are "small" values such as single quadruple addresses.)

8. (3p) **Error Handling**

   Explain, define, and give examples of using the following concepts regarding error handling:

   (a) (1p) Valid prefix property,

   (b) (1p) Phrase level recovery,

   (c) (1p) Global correction.

9. (3p) **Memory management**

   (a) (1p) Non-local references: How does a static link work?

   (b) (1p) Non-local references: How does a display work?

   (c) (1p) Dynamic data: How is the actual size and contents of a dynamic array handled?

10. (6p) **Intermediate Code Generation**

    (a) (3p) Given the following code segment in a Pascal-like language:

```
if x=y
  then x:=x-10
  else while y>10 do
    if y<x
      then y:=y+1
      else y:=func(x)
```

    Translate the code segment into an abtract syntax tree, quadruples, and postfix code.

    (b) (3p) Divide the following code inte basic blocks, draw a control flow graph, and show as well as motivate the existing loop(s):

```
    goto L2
L1: x:=x+1
L2: x:=x+1
    x:=x+1
    if x=1 then goto L1
L3: if x=2 then goto L4
    goto L5
L4: x:=x+1
L5: x:=x+1
    if x=4 then goto L3
```

11. (TDDB44 only – 6p) **Code Generation for RISC, etc.**

    (a) (1.5p) Explain the main similarity and the main difference between superscalar and VLIW architectures from a compiler's point of view. Which one is harder to generate code for, and why?

    (b) (1.5p) Explain briefly the concept of software pipelining. Show it with a simple example.

    (c) (2p) What is branch prediction and when is it used? Why is this important for pipelined processors?

    (d) (1p) What is a live range? Explain the concept and show a simple example.

12. (TDDB44 only – 3p) **Compiler Lab Exercises**

    Points are fetched from LADOK and granted automatically provided the labs were finished on time. Only write something in this assignment if you believe you should have been reported in LADOK but was not registered yet for some reason.